AD-A060 719 WISCONSIN UNIV-MADISON MATHEMATICS RESEARCH CENTER F/G 9/2

LANGUAGE REQUIREMENTS AND DESIGNS TO AID DATA ANALYSIS AND STAT--ETC(U)

JUN 78 G N WILKINSON DAAG29-75-C-0024

UNCLASSIFIED MRC-TSR-1857 NL

| OF |

AD
A060719

END
DATE
FILMED
01 -79
DDC

MRC Technical Summary Report #1857

LANGUAGE REQUIREMENTS AND DESIGNS TO AID DATA ANALYSIS AND STATISTICAL COMPUTING

Graham N. Wilkinson

Mathematics Research Center
University of Wisconsin–Madison
610 Walnut Street
Madison, Wisconsin 53706

June 1978

(Received June 6, 1978)

Approved for public release
Distribution unlimited

Sponsored by

U.S. Army Research Office
P.O. Box 12211
Research Triangle Park
North Carolina 27709

UNIVERSITY OF WISCONSIN - MADISON
MATHEMATICS RESEARCH CENTER

6 LANGUAGE REQUIREMENTS AND DESIGNS TO AID DATA ANALYSIS AND STATISTICAL COMPUTING

10 Graham N. Wilkinson

11 Jun 78

9 Technical Summary Report #1857
~~June 1978~~

ABSTRACT 14 MRC-TSR-1857

An outline is given of design requirements for high-level statistical computing languages, with special emphasis on nameable data-structures, general operations including matrix and table calculations, looping, branching and the use of macros, and on user-extendability. Implementation and evaluation are discussed briefly.

15 DAAG29-75-C-0024

12 27p.

AMS(MOS) Subject Classification - 62-02, 68-00, 68A05, 68A30

Key Words - Statistical computing languages, compiler-compilers, data structures, extendability, GENSTAT system, macros

Work Unit Number 4 - Probability, Statistics and Combinatorics

Sponsored by the United States Army under Contract No. DAAG29-75-C-0024.

221 200

SIGNIFICANCE AND EXPLANATION

This report comprises an invited address to the International Statistical Institute at the 41st Session in West Delhi, India, December 1977, on the occasion of the inauguration of the new International Association for Statistical Computing.

In statistical data processing centers such as statistical departments of research institutes or government statistical offices, the major processing cost is in the preparation of computing tasks for the computer, particularly, when low-level languages like Fortran are used. Typically the preparation and programming costs exceeds the actual computing charges by a factor of something like ten or more, and these costs can be dramatically reduced by as much as 90% if sufficiently powerful high-level problem-oriented programming languages are available, for example the Genstat system in England and Australia, which accounts for a substantial fraction of all statistical computing in those countries. The paper addresses itself to the general design requirements for such high-level languages and their implementation.

The responsibility for the wording and views expressed in this descriptive summary lies with MRC, and not with the author of this report.

LANGUAGE REQUIREMENTS AND DESIGNS TO AID DATA ANALYSIS AND STATISTICAL COMPUTING

Graham N. Wilkinson

Mathematics Research Center
University of Wisconsin-Madison
Madison, Wisconsin 53706 USA

1. INTRODUCTION

The inauguration of the International Association for Statistical Computing at this meeting of the International Statistical Institute in India is an appropriate occasion for reviewing developments in statistical computing and for considering where we might be heading. My role in this paper is to discuss the need and development of high-level languages for statistical computing, though I should stress that my competence to do this is limited, as my own experience relates primarily to statistics in scientific research and not to the official and commercial branches of statistics with which many of you are concerned. I think nevertheless that my remarks can be regarded as applicable to any area where statistical computations are necessary for summarizing and interpreting observational data. However, for reasons of ignorance, I shall not be considering an important complementary area of language development, namely that for database management systems which are needed for generating, validating and interrogating large data banks.

Sponsored by the United States Army under Contract No. DAAG29-75-C-0024.

The paper is addressed primarily to statisticians rather than computer specialists. Illustrations of some of the points will be drawn mainly from the GENSTAT Statistical Computing System (Nelder, et al., 1975), to whose development I contributed for several years (1966-75). However, this is just one of a number of major statistical computing systems that are being widely used today, and I shall also be referring briefly to BMD (Dixon (ed.), 1975), PSTAT (Buhler and Buhler, 1976), SAS (Barr, et al., 1976), SPSS (Nie, et al., 1975), and RGSP (Yates, 1975).

2. WHY A STATISTICAL COMPUTING LANGUAGE?

I think statisticians have a right to expect to be able to describe the data for a statistical analysis, and the statistical and other operations to be performed on those data, in a clear, concise and intelligible form, in statements (directives) which are free from extraneous technical details that relate only to computer implementation of the task. Also, in view of the very great variety of statistical and other operations that may be needed, they have a right to expect that the method of describing a computing task should not have to change arbitrarily for each different operation, that is, a highly consistent syntax is required, together with maximum flexibility in linking together various operations in an order appropriate to the problem. One requires also that the results of the various analyses should be presented with a consistency of format and with annotation sufficient to render the results readily intelligible to all who might use them. Finally of course, one does not want to have to read 30-50 shelf-centimeters of computer documentation to find out how to accomplish one's computing tasks.

These requirements can only be met fully with some small number of high-level statistical computing languages, each properly supported by an interpretive computing system of high quality, and adapted to various needs such as large-scale batch-processing of data or high-speed interactive analysis of small

data sets at computer terminals. In the present state of the art it seems unlikely that a single system would meet all needs.

A well-designed language simplifies the task of providing user documentation in the form of user manuals and examples. It is also the right organizational key to properly modularized computer implementation. The more general and flexible the language the greater will be its frequency of use, and hence the greater the economic and professional incentives for refining and extending the system and for achieving portability over a range of different computing environments.

Let me illustrate the point about conciseness, intelligibility and absence of technical jargon with a simple GENSTAT program for analysing a randomized Block experiment, taken from a talk I gave earlier at the Ninth Interface Symposium on Computer Science and Statistics last year:

```
'REFERENCE'   GNWAOV1
"RANDOMIZED BLOCK EXPERIMENT TESTING 4 WHEAT VARIETIES"
'UNITS'       $20
'NAMES'       VN = GABO, PRIOR, SA1, Q53   "VARIETY NAMES"
'FACTORS'     BLOCK $5 : PLOT $4 : VARIETY $VN
'GENERATE'    BLOCK, PLOT
'READ/P'      VARIETY, YIELD
'BLOCK STRUCTURE'    BLOCK/PLOT    'TREATMENT'   VARIETY
'ANOVA'       YIELD
'RUN'
     (data records inserted here)
'CLOSE'       'STOP'
```

The program is almost self-explanatory. It comprises a number of statements (directives), some of which define data structures or model formulae and others the operations required. System words are identified with apostrophes, comments with double apostrophes. The program begins with a reference name and a brief descriptive comment. Next are specified the number of

experimental units (20) and the names of the varieties in the experiment. The FACTORS statement names the three factors with respect to which the data are classifiable, together with either the number of categories in each case or else, for variety, a reference (vn) to the relevant list of category names. The next, GENERATE directive says more about the tabular structure of the data, specifically that the 20 experimental units in serial order of reference form a two-way (5×4) table with respect to block and plot, with the plot index changing most rapidly. This directive will cause the following index lists for block and plot to be created:

Unit number	1	2	3	4	5	6	. . . . .	19	20
Block index	1	1	1	1	2	2	. . . . .	5	5
Plot index	1	2	3	4	1	2	. . . . .	3	4

We still don't know how varieties were assigned to the plots but the next READ directive indicates that this varietal information is specified in parallel as part of the data file to be input, the other part being an observational variable named yield.

When the program is executed to this stage, a data matrix with 20 rows and 4 columns (block, plot, variety, yield) will have been defined. The next step is to specify symbolically a model for the required analysis of variance. The BLOCK directive describes the error component of the model as a nested structure of the form $b_i + (bp)_{ij}$, $i = 1\cdots5$, $j = 1\cdots4$, and the treatment component of the model is specified separately by a TREATMENT directive. The ANOVA directive will then analyse the variate yield accordingly. The remaining directives (RUN, CLOSE, STOP) bring about execution of the job and termination of the computer run. In practice several jobs, each beginning with a REFERENCE name and ending with CLOSE, can be executed in one run.

One can of course envisage different and equally valid ways of expressing the information in this program, but the essential point is that no simpler description of the data and the analysis

is really possible if intelligibility is to be preserved. Note that the names introduced by the user typically serve two functions, first as program identifiers essential to the job description, and second as labelling information for annotating output results.

Though the language requirement for this example appears fairly trivial, in practice one may also wish to undertake a whole variety of analysis steps, such as checking for outliers, graphing residuals against plot position, covariance adjustment for an initial fertility variable, testing the effect of different transformations and so forth. And with more complex, multivariate data sets the variety of analysis steps multiplies considerably. To have the flexibility to do [illegible] with a full and expanding set of statistical operations, t[illegible]to account the tremendous variety of structure in and relevan[illegible]dels for the data sets, one needs languages of considerable power both in their descriptive capabilities as well as in the range of statistical and other operations provided in their implementation. I shall discuss some of the language requirements in the subsequent sections of this paper.

3. THE OPERANDS OF A STATISTICAL COMPUTING LANGUAGE

To attain the required degree of flexibility and descriptive power the language must provide for naming and specifying a wide range of basic data structures and other entities. It must also be possible for arbitrary collections of such structures to coexist in central memory. This implies that the interpretive system must have a reasonably sophisticated subsystem for data management, with a names directory pointing to attribute lists, each specifying the essential characteristics of the corresponding data structure and the relative addresses of its list or lists of values. Generally, more complex structures will point to

simpler constituent structures. For instance a factor may point to a level-names vector, and a multi-way table will point to the factors which define it.

Of large-scale systems currently in use, GENSTAT appears to be the most ambitious in this regard. The nameable numeric data structures include scalars, vectors, matrices (distinguishing between rectangular, square symmetric and diagonal matrices), factors, multiway tables and SSP structures for regression and multivariate analysis. One may also name text (headings), arrays of names, pointer vectors defining a collection of data structures, model formulae, macros, and subfiles and files of data structures for storage and retrieval.

For a comparison of several major statistical systems with regard to their data-structure operands, see Payne and Nelder (1976).

The question naturally arises as to how much more generality needs to be provided for defining data structures. Every new form of structure adds to the burden of the data-management subsystem and might sometimes actually complicate the writing of a statistical program, since most of the operations available in the language will be limited to the simpler forms of operand. I think a distinction has to be made between what may be termed the standard structures in the language which are fully supported by the interpretive system as a whole, and special structures which are intelligible to perhaps only a few inter-related directives in the language, and for which complete protection from corruption (by other directives) cannot be guaranteed (chiefly because a special structure may point to constituent standard structures which are freely modifiable by other directives). This is the policy which has been followed in the GENSTAT implementation.

4. OPERATIONS

Any statistical computing language will have special directives for analysis of variance, survey analysis, regression, non-linear estimation, multivariate analysis etc. But if the user is not to be inhibited from devising his own special forms of analysis, the language must also provide a wide range of general operations on the standard structures.

Thus a full vector and matrix calculus is essential, and should also extend to multi-way tables, since these are basic operands for many statistical operations. In Genstat the CALCULATE directive will accept and evaluate any interpretable algebraic expression comprising any of the standard operands together with the standard arithmetic and logical operators as defined in Fortran, as well as a wide range of mathematical and statistical functions. The CALCULATE directive also accepts lists of data structures as operands as in

```
'CALC'  LOGHT,LOGWT = LOG(HT,WT)
```

which is equivalent to

```
'CALC'  LOGHT = LOG(HT) : LOGWT = LOG(WT) .
```

There must also be general READ and PRINT directives that will handle any of the standard data structures, and of course flexible file storage and retrieval directives, since multi-stage processing is common in statistical data analysis.

Branching and looping facilities are essential for specifying iterative forms of analysis, and analyses in which some steps are contingent on the outcome of others. The FOR-loop syntax devised by J. A. Nelder for Genstat is of particular interest. The basic form for looping over corresponding variables in say two parallel lists of variables is illustrated by

```
'FOR'  X = xlist;  Y = ylist
[directives specifying operations on  X,Y]
'REPEAT' .
```

Here the symbols X and Y are loop dummies which in each pass through the loop are set to refer to the next pair of variables from the specified variable lists xlist and ylist respectively. The important feature of the syntax is that it circumvents the need to specify the lists explicitly as indexed arrays of variables, as would be the case if a Fortran-like loop syntax were used. The FOR-loop continues until the longest list is exhausted, shorter parallel lists being recycled until the loop terminates. Nested loops are allowed.

The Genstat branching directives are also of interest in that one directive suffices for both conditional and unconditional branching. With an entry point in the program specified with a LABEL directive, e.g. 'LABEL' PROBCALC, the branching directive could be 'GOTO' PROBCALC or 'GOTO' PROBCALC*(N.GT.0). Genstat also provides an ASSIGN directive with the syntax 'ASSIGN' dummy = list $ index, which sets the dummy variable to refer to the variable in the list indicated by the current value of index. If the list comprises labels, this directive acts like a Fortran ASSIGN.

A facility for defining and calling macros is essential for enabling the language is to be used to define new statistical procedures in terms of the operations already provided in the language and more generally in terms of other macros that have been developed. A macro is a named block of directives. In Genstat the definitional syntax is 'MACRO' macroname $ set of directives 'ENDMAC', and a macro can be invoked in a program with the directive 'USE' macroname. Macros may use other macros. In an interpretive language a macro is the analogue of a subroutine in FORTRAN with the difference that a macro is stored and retrieved only in source-language form rather than in compiled form. Compilation (translation to object code) takes place only when required, as a part of the program which uses it.

This feature gives added flexibility, enabling the compiler to take context-dependent action when necessary, as well as simplifying implementation.

A hopeless confusion of identifier names could arise in building a macro library unless the scope of the identifiers is properly limited. For instance does the identifier X in a macro refer to the same variable as an identifier X in the calling program, or not? Thus one must distinguish between local variables, common variables and dummy variables for external parameters, as in Fortran. Genstat provides a directive LOCAL for defining local variables in a macro, but I would prefer to see instead a COMMON directive, and also the macro definition extended to allow a list of local dummy arguments for external variables to be specified, as allowed in PSTAT.

For really sophisticated uses of a language as powerful as Genstat it becomes important to allow almost arbitrary user-controlled alternations between compiling and executing actions. For instance, it is possible in Genstat to postpone compilation of a macro until execution time (run time) with the modified directive 'USE/R' macroname, if it is the case that preceding directives must be executed to define some of the entities that the compiler would reference when compiling the macro.

For full flexibility in writing macros, almost all entities in the language should be specifiable with identifiers, that is, there should be almost no obligatory constants. It is also important that accessing directives be provided to allow the user to refer to various parts of special output structures that a directive like ANOVA might create.

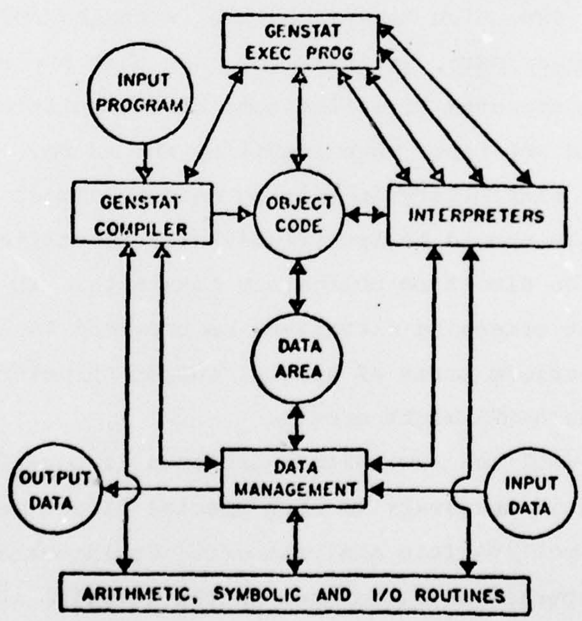
One may well ask now, with a language as powerful as described above, why is it necessary to have special directives for analysis of variance, multivariate analysis etc. Could not all complex statistical operations be defined in the language itself using the macro facility, rather than by adding more Fortran code say, to the interpretive system. However if one looks at the Fortran

source code of a large system, one sees a mass of computing detail and decision logic which could not be implemented efficiently in macro form in the higher level language. Typically many thousands of Fortran statements are involved, and one of the most critical aspects of designing a higher level language is to ensure that the range of facilities provided is sufficient to avoid inefficient execution in most circumstances.

5. IMPLEMENTATION

The importance of the language approach is that it creates the right kind of design philosophy in building an interpretive system. Pitfalls which might otherwise block the further development of the system can be more readily envisaged and avoided.

The interpretive system is typically a single compiled program written in a base language such as Fortran, because this is the easiest and most logical way of implementing a higher level language. The various components of an interpretive system are illustrated in the following diagram taken from



Wilkinson (1976). There is a compiler which reads the user's program and translates it into object code (not necessarily in one step). The executive program scans this code at execution time and calls in the corresponding interpretive routines which carry out the operations required, referencing relevant data through a data management subsystem. At the base of the system is a calculus of elementary subroutines to perform the various arithmetic, symbolic and input/output operations. Typically there are several hundred such subroutines.

Perhaps the easiest component of the system to develop (with the hindsight now available) is the language compiler (object code generator), for this is an area where computing science software technology is very well developed. Essentially the compiler contains, in symbolic tabular form, a complete description of the language and its grammar (syntax). This tabular information controls the action of a parsing algorithm which interprets any input sentence written in the language, checks it for syntactic errors, and makes the appropriate translation into the internal object code. Very good fast parsing algorithms have been available for some time (Aho and Ullman, 1972-73). The system designer can also make use of what is termed a compiler-compiler. This is a text processor which will generate a compiler for the language from an input language description and an input set of the corresponding translating actions required when input sentences are interpreted (Johnson, 1976). An important ancillary function of the compiler-compiler is to verify that the language input to it is in fact syntactically self-consistent, that is, free of possible parsing ambiguities. Disambiguating rules can be adjoined to the language to remedy such defects.

At the other extreme, the hardest part of the system to design and build is the set of interpretive routines and the supporting calculus of elementary subroutines, for here we are dealing with the semantics (meaning) of the language rather than its syntax. These routines constitute in fact the bulk of the

system. The overall design requirement here is to add no more source code to the system than is absolutely necessary at any stage of development, and this necessitates in practice a continuing process of refining the existing calculus of routines to enhance as much as possible the computing and manipulative power of the calculus. Without such discipline in development, a statistical computing system could rapidly grow to an unmanageable size. Some statistical computing systems are probably larger than they ought to be, in relation to the facilities they provide. Better use could be made of macros in the higher level language, though many systems do not currently have this feature.

Portability is no longer the issue it used to be a decade ago. Most major statistical computing systems are now supported on a range of different computers, the notable exception being SAS which is very much tied to the IBM range. SPSS is supported by some twenty conversions on different computers. PSTAT and GENSTAT have followed the policy of maintaining a master source file for all versions from which source for a particular version can be created by a text processor.

6. EXTENDABILITY

The capability for user-defined extensions to a statistical computing system would be an important requirement for many statistical data-processing centres, particularly if new methods of analysis are being developed. If the high-level language is powerful enough, the macro facility should be sufficient to cope with special processing tasks of a transient nature, for which a possible computing inefficiency is not significant in the context of the whole job mix. However there certainly will be instances where the user would want to extend the system at the base language level, for instance to add Fortran routines for time series analysis to a system lacking special directives in this area.

There is no difficulty about this in principle, provided that the operating system for the particular computing centre has incremental link-editing facilities. For then additional Fortran can be added to a variable user-overlay which would be subsequently recompiled and linked again to the statistical system. PSTAT, SAS and GENSTAT all provide for user extensions, though GENSTAT's facilities are not yet in the public domain I believe, pending development of appropriate documentation.

The high level language must include a special communicating and parameter-passing directive, which both causes the executive program to call in the user-defined overlay, and transmits a pointer to the relevant parameter list of data structures in object-code form. GENSTAT for instance has a directive OWN with the syntax 'OWN' parameter-list. More generally one would also want a branch parameter as in 'OWN/branch' parameter-list, to allow several users' sub-overlays to co-exist. More sophisticated forms are feasible if the system has a good table-driven compiler, for then one could reprocess the compiler to accept a user-specified directive name and even a user-specified syntax for the argument list of the directive. In other words, one could also extend the language itself, as well as add the necessary additional routines to the interpretive system.

With a well designed system only a minimal amount of information about it should be needed for user-defined extensions. The programmer would need to know, in particular, about the routines for accessing data structures and for workspace management. In the case of GENSTAT there is one main data-structure accessing function, which operates on common arrays of structure attributes.

It should be stressed that if facilities for user-defined extensions are available, the system's customers will very likely contribute substantially to its further development. SAS has

benefited particularly in this way. At the Bell Laboratories in Murray Hill, New Jersey, a language 'S' is being developed with extendability incorporated at the outset as an essential design feature (J. M. Chambers, personal communication).

7. EVALUATION

What advice should be given to the director of a major centre for statistical analysis and data-processing, someone who is most likely not a computer specialist, but who must from time to time re-evaluate the needs of his centre in regard to statistical computation?

Clearly I think he should consider organizing the bulk of his centre's statistical computing through one or more powerful high-level statistical languages, and that in selecting from available systems he should place considerable weight on the extendability of the supporting systems as well as on the support provided by their originating centres in terms of documentation, portability, counselling services, further development etc. With more than one system the transferability of data files between them will be an important consideration.

At a centre which currently does most of its computing via ad hoc Fortran (or PL/I etc.) programming, some 30% or more of total staff effort may be tied up in this activity. In terms of salary and overheads these programming costs may well exceed actual computing charges by a factor of 10 or more, and could well be reduced by as much as 90% with the use of high-level statistical languages, thereby releasing valuable staff resources for other activities.

Use of high-level languages also increases the productivity of a computer installation in terms of scientific data analysis output. Often as much as 50% of computer activity is tied up in compiling and debugging Fortran programs, and this figure can be reduced to as low as a few percent, thereby nearly doubling productivity, if the users make extensive use of high level

languages. This has been the experience with GENSTAT at the Rothamsted computing centre, which services some 36 agricultural research institutes in Britain. Genstat now accounts for about 75% of the total statistical throughput, and most of the remainder is handled by other general programs such as the survey analysis system RGSP (Yates, 1975). In Australia, Genstat, in competition with other statistical computing systems such as SPSS and BMD, has come to be the preferred system for statistical processing on the CSIRO computing network CSIRONET. It processes some 700 to 800 jobs a week, and is used several times more frequently than any of the other available systems or packages (R. I. Baxter, personal communication). An extensive Genstat macrolibrary has been developed there.

In view of the current interest in mini-computers it is important to stress that powerful high-level languages of the kind I have been describing require large computers for efficient implementation. Something like 200,000 bytes of central memory and considerable back-up memory is a normal requirement. The director of a statistical centre may of course receive conflicting advice about this, particularly from his own programming staff, who may have a vested personal interest in maximum exploitation of mini-computers, regardless of the programming effort required. Careful cost-benefit analysis of alternative proposals is therefore needed, and proper weight should be given to the opinions of experts in computer science, particularly in the area of language implementation. Mini-computers do have a definable auxiliary role to play other than as terminal components of a larger computer system, for instance in ad hoc small-scale computations, primarily mathematical in content, for which the data-management requirements are trivial enough not to require a programmed management system. (The user himself keeps track of what and where in memory his data entities are, etc.)

Centres which are contemplating building their own high-level language and interpretive system should note the very substantial investment of effort that may be required, as much as 30 man-years for a really powerful, fully documented system, even when extensive use is made of available subroutine libraries.

It should be noted, too, that the efficiency of implementation of a high-level language may depend critically on the computer environment, particularly on the supporting system software. In Australia, where the central computer of CSIRONET is a CDC CYBER 76, the average cost of a Genstat job is about $1.30, whereas on a different computing installation in England the average cost for much the same job-mix is several times higher, due mainly to a very inefficient file-management system having a critical effect on the running of heavily overlaid programs (which most interpretive systems are).

Objective evaluation studies of the various statistical computing systems currently in use have an important role to play, not only to provide guidance for prospective customers of such systems, but also to stimulate improvements in the design and implementation of high-level languages. For some useful discussion and references see Francis and Sedransk (1976). The American Statistical Association, in particular, is to be commended for the valuable evaluation studies initiated by its Statistical Computing Section, and it is hoped that the newly-formed International Association for Statistical Computing will contribute to a world-wide improvement in knowledge with the proposed Technical Information Exchange on Statistical Software (Muller, 1977).

BIBLIOGRAPHY

Aho, A. V. and Ullman, J. D. (1972) Theory of Parsing, Translation and Compiling, Vols. 1 (1972) and 2 (1973). Prentice-Hall, New Jersey.

Barr, A. J., et al. (1976) A User's Guide to SAS76. SAS Institute, Raleigh, North Carolina.

Buhler, S. and Buhler, R. (1976) PSTAT Reference Manual (Version 3.07). Princeton University Computer Center.

Dixon, W. J. (ed). (1975) Biomedical Computer Programs. University of California Press.

Francis, I. and Sedransk, J. (1976) Software Requirements for Analysis of Surveys. Proc. Ninth International Biometrics Conference, Vol. 2, 228-252.

Johnson, S. C. (1976) Compiler-compilers: Where have we been and where are we going. Proc. Ninth Interface Symposium on Computer Science and Statistics, 56-58.

Muller, M. E. (1977) IASC: Purpose and Aspirations. (To appear in the Proceedings of the Forty-First Session of the International Statistical Institute, New Delhi, India.)

Nelder, J. A., et al. (1975) GENSTAT Reference Manual. Program Library Unit, University of Edinburgh.

Nie, N. H., et al. (1975) SPSS. McGraw Hill, New York.

Payne, R. and Nelder, J. A. (1976) Data Structures in Statistical Computing. Proc. Ninth International Biometrics Conference, Vol. 2, 191-208.

Wilkinson, G. N. (1976) The Language Approach to Statistical Computing. Proc. Ninth Interface Symposium on Computer Science and Statistics, 71-73.

Yates, F. (1975) The Rothamsted General Survey Program (Parts 1 and 2). Program Library Unit, University of Edinburgh.

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
1857

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
LANGUAGE REQUIREMENTS AND DESIGNS TO AID DATA ANALYSIS AND STATISTICAL COMPUTING

5. TYPE OF REPORT & PERIOD COVERED
Summary Report - no specific reporting period

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Graham N. Wilkinson

8. CONTRACT OR GRANT NUMBER(s)
DAAG29-75-C-0024

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Mathematics Research Center, University of Wisconsin
610 Walnut Street
Madison, Wisconsin 53706

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
Work Unit Number 4 - Probability, Statistics and Combinatorics

11. CONTROLLING OFFICE NAME AND ADDRESS
U. S. Army Research Office
P.O. Box 12211
Research Triangle Park, North Carolina 27709

12. REPORT DATE
June 1978

13. NUMBER OF PAGES
17

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15. SECURITY CLASS. (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Statistical computing languages, compiler-compilers, data structures, extendability, GENSTAT system, macros

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
An outline is given of design requirements for high-level statistical computing languages, with special emphasis on nameable data-structures, general operations including matrix and table calculations, looping, branching and the use of macros, and on user-extendability. Implementation and evaluation are discussed briefly.